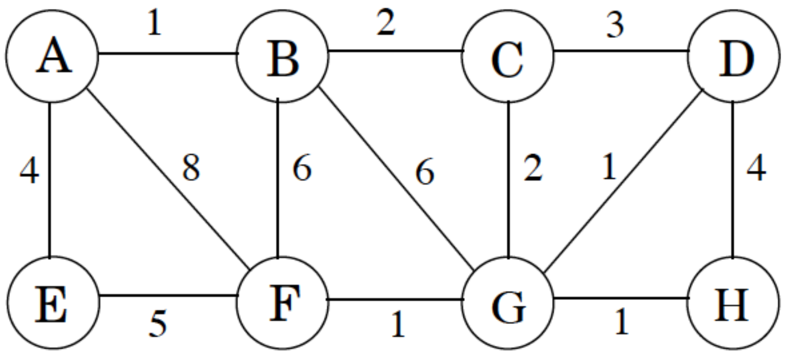
CMPSC 465 Data Structures & Algorithms
Spring 2024 Mehrdad Mahdavi and David Koslicki

Worksheet 9

Wednesday, Mar 20, 2024

1. **Minimum Spanning Trees.** Run Prim's Algorithm to find a minimum spanning tree for the following graph. Whenever there is a choice of nodes, always use alphabetic ordering (e.g. start from node A). Show the order edges are added and the weight of the partial MST at each step.



Solution:

Vertex included	Edge included	Cost
A		0
B	AB	1
C	BC	3
G	CG	5
D	GD	6
F	GF	7
H	GH	8
E	AE	12

2. **Edge Weight Incrementing.** Consider an undirected graph $G = (V, E)$ with nonnegative edge weights $w_e \geq 0$. Suppose that you have computed a minimum spanning tree of G, and that you have also computed shortest paths to all nodes from a particular node $s \in V$.
Now suppose each edge weight is increased by 1: the new weights are $w'_e = w_e + 1$.

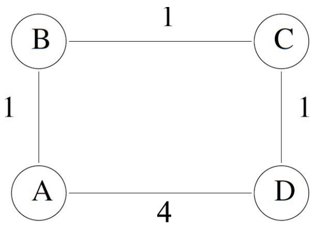
   (a) Does the minimum spanning tree change? Give an example where it changes or prove it cannot change.

   (b) Do the shortest paths change? Give an example where they change or prove they cannot change.

Solution:

(a) The minimum spanning tree does not change. Since, each spanning tree contains exactly $n-1$ edges, the cost of each tree is increased $n-1$ and hence the minimum is unchanged.

(b) The shortest paths may change because the number of edges in two competing shortest paths may be different. In the following graph, the shortest path from A to D changes from $AB-BC-CD$ to AD if each edge weight is increased by 1.



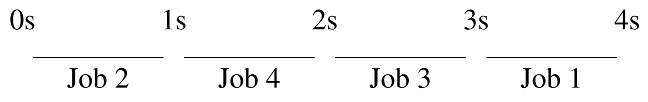
3. **Minimum Spanning Trees and Subgraphs.** Let T be an MST of graph G. Given a connected subgraph H of G, show that $T \cap H$ is contained in some MST of H.

Solution: Let $T \cap H = \{e_1, \ldots, e_k\}$. We use the cut property repeatedly to show that there exists an MST of H containing $T \cap H$.

Suppose for $i < k$, $X = \{e_1, \ldots, e_i\}$ is contained in some MST of H. Removing the edge e_{i+1} from T divides T in two parts giving a cut $(S, G \backslash S)$ in G and a corresponding cut $(S_1, H \backslash S_1)$ of H with $S_1 = S \cap H$. Now, e_{i+1} must be the lightest edge in G (and hence also in H) crossing both cuts. If this were not true, we could include the lightest edge and remove e_{i+1}, creating a tree lighter than T and contradicting our assumption that T is an MST. Also, no other edges in T, and hence also in X, cross this cut, since T is a tree. We can then apply the cut property to get that $X \cup e_{i+1}$ must be contained in some MST of H. Continuing in this manner, we get the result for $T \cap H = \{e_1, \ldots, e_k\}$.

4. **Job Scheduling.** You are given a set of n jobs to run on a computer in the next T seconds. Each job takes one second to run, and the computer runs one job at a time. Job i has an integer deadline $0 \leq d_i \leq T$ and a penalty $p_i \geq 0$. If job i isn't finished by d_i, you'll have to pay a penalty of p_i dollars. Jobs may be scheduled to start at any non-negative integer time. The goal is to schedule all jobs so as to minimize the total penalty incurred.

Penalty p_i	1	2	5	4
Deadline d_i	1	1	3	3
i	1	2	3	4



For example, suppose we have four jobs, with penalties and deadlines as shown on the left side of the figure above. If we schedule them as shown on the right, then we incur a penalty of 1 because Job 1 did not finish before its deadline $d_1 = 1$.

For each of the following greedy algorithms, either prove that it is correct, or give a simple counterexample (with at most three jobs) to show that it fails.

(a) Among unscheduled jobs that can be scheduled on time, consider the one whose deadline is the earliest (breaking ties with the highest penalty), and schedule it at the earliest available time. Repeat.

(b) Among unscheduled jobs that can be scheduled on time, consider the one whose penalty is the highest (breaking ties with the earliest deadline), and schedule it at the earliest available time. Repeat.

(c) Among unscheduled jobs that can be scheduled on time, consider the one whose penalty is the highest (breaking ties arbitrarily), and schedule it at the latest available time before its deadline. Repeat.

Solution:

(a) Counterexample: Consider a listing of (d_i, p_i) as $(1,1),(2,10),(2,20)$. The algorithm schedules $(1,1)$ at time 0, $(2,20)$ at time 1, and $(2,10)$ late, incurring a penalty of 10. One better (actually optimal) scheduling is to schedule $(2,10)$ at time 0, $(2,20)$ at time 1, and $(1,1)$ late, incurring a penalty of 1.

(b) Counterexample: Consider a listing of (d_i, p_i) as $(1,1),(2,10)$. The algorithm schedules $(2,10)$ at time 0, and $(1,1)$ late, incurring a penalty of 1. One better (actually optimal) scheduling is to schedule $(1,1)$ at time 0, $(2,10)$ at time 1, incurring no penalty.

(c) Proof of Correctness: Let S be an optimal schedule of jobs and let G be the schedule of jobs generated by this greedy algorithm.

Assume $G \neq S$ (otherwise greedy is trivially optimal), and imagine that we run the greedy algorithm: since $G \neq S$, the greedy algorithm must eventually make a scheduling choice that does not coincide with S. Specifically, we will analyze a schedule S', which makes the choice to schedule job j_k at time t. Let j_l be the job scheduled by S at time t if such a job exists. (That is, the greedy algorithm scheduled a bunch of jobs, but they were all scheduled at the same times as in S, until it scheduled j_l.)

We claim $p(S') \leq p(S)$ (the penalty paid by S' is no more than that paid by S) and therefore S' must also be optimal. There are a few cases:

- *S did not schedule a job at time t.* In this case, the only job that moved was j_k. The only way S' can be worse than S is if j_k is scheduled late in S'; however, by the definition of the greedy algorithm, j_k was scheduled before its deadline, so $p(S') \leq p(S)$.
- *Swapping j_k and j_l moves job j_k later in the schedule.* By definition of G, we know that j_k was scheduled before its deadline, so S' does not pay a penalty for j_k. Likewise, the swap moved j_l earlier in the schedule, so S' can only pay a penalty for j_l if S paid the same penalty, and thus $p(S') \leq p(S)$.
- *Swapping j_k and j_l moves job j_k earlier in the schedule.* First, we show that S must have scheduled j_k late. Note that all time slots in the range $\{t+1, \ldots, d_k - 1\}$ (i.e. slots after t but before j_k's deadline) have already been assigned in G, otherwise greedy would have scheduled j_k there instead of at time t. However, we know that all jobs that have already been scheduled match S, and therefore S must schedule j_k late (after d_k).
  Second, we observe that greedy has not scheduled j_l and therefore $p_k \geq p_l$ because it chose to schedule j_k first.
  Putting these together, we know that j_k was late in S and on time in S', so the worst-case scenario for S' is that j_l was on time in S and late in S'. Thus, we get

$$p(S') \leq p(S) - p_k + p_l \leq p(S) \ .$$

Thus, $p(S') \leq p(S)$, but since $p(S)$ is optimal it follows that $p(S') = p(S)$ and S' is also optimal.

Finally, notice that if greedy agreed with S for m steps (i.e. it scheduled job j_k on step $m+1$), it will agree with S' for at least $m+1$ steps. Thus, we apply the above argument to S' to get S'' and so on. Eventually, we get an optimal schedule S^e with which greedy agrees for all jobs that were not late, implying G is optimal (all other jobs pay the penalty regardless of when they are scheduled).